CIS 111 WEB PROGRAMMING

Project 1

Due Date

23:59 F 6/29 (XC)
23:59 M 7/2 (FC)

EXTRA CREDIT FOR SPEEDY DELIVERY

Students who complete this project by 6/29 will receive +5 XC points added to this project score.

Project 1 may be submitted by 7/2 for full credit (FC).

This is the only project that will have two due-dates.

CIS 111 Web Development w/JavaScript

CIS 111 Web Development

Project 1

PURPOSE

CIS 111 projects and labs provide hands-on practice in coding and web development using JavaScript, HTML, and CSS.

In the 21st century, every business– from fashion to finance– is a tech company. Every company has a web presence, and many have smart phone apps. Coding and web dev, therefore, are high-demand skills for every job, and differentiate you from most people in the job market. All employers rank STEM skills very highly.

By completing this project you will practice skills that are essential not only to your success in this course, but will also contribute to your academic success at the UO, and in your professional life after graduation.

Programming improves your logical thinking skills– it's is analogous to constructing proofs in mathematics. Programming serves the same role that Latin has for centuries-- it sharpens your intellect.

CIS 111

Project Requirements

WHY LEARN TO JAVASCRIPT?

With the rise of the web as a platform for running applications (desktop and mobile) JavaScript is one of the most widely used programming languages on the planet.

When you open Facebook or Gmail, your browser spends more time processing JavaScript than it does rendering HTML and CSS. JavaScript is now used to power web servers as well, in the form of Node.js.

JavaScript is the native language of the web platform. Since it is built-in to every browser, no download or installation is required–you can start programming JavaScript immediately.

In 111, you will build cross-platform, mobile-friendly web applications using technologies and workflow tools used by professional developers.

LEARNING OUTCOMES

CIS 111 includes, an understanding of the foundational ***Concepts*** of programming, a ***Skills*** component, and the ability to use problem-solving intellectual ***Capabilities*** in an information technology context:

A) CONCEPTS (KNOWLEDGE**):** The purpose of this project is to help you to become familiar with the following important condepts:

- Command-Line JavaScript
- Basic JavaScript statements, expressions and variables

B) SKILLS: This project will also to help you practice the following skills that are essential to your success in this course:

- The four steps of the 110 WebDev Workflow (see the Glossary for details). The sooner you understand and master this

workflow, the sooner you will start getting full credit on your projects.

C) CAPABILITIES: This project will help you learn to *read and understand* the project software requirements.

You have understood a project requirement when you know: (A) What to Do and, (B) How to Do It.

- Do the required readings in our textbook *JABG* before starting this project.
- Ask questions in class in lab, when requirements are not clear.
- Get started on this project early, without delay.
- Budget time for Help Hours visits. You will not always be able to complete a project w/o assistance. Therefore, get started early to make sure you can get the help you need when you need it.
- Solve each project requirement in order by creating a web page. This requires you to apply the four steps of the 110 WebDev Workflow (see the Glossary for details).

JAVASCRIPT PROGRAMMING TIPS

A) *Always* declare your variables by using the *var* or *let* keywords, as follows:

```
var num1;
var num2 = 0, firstName = "Anon";
let fName = prompt("Enter first name");
```

B) Always make this the first line of code in your .js files in Atom:

```
// jshint esversion: 6
```

C) Read Beware the Browser Cache.

PROJECT REQUIREMENTS

For best results, solve the following problems in order.

This project will be discussed in detail in class; that is the best place to ask questions about the project.

1. **[25 pts] hello-world.html**.

   Create your 111 Webiste is the document that covered the week 1 lab agenda, and is available in Canvas.

   The document includes a section, *Hello, World in JavaScript*.

   Complete all four steps in this section (Edit/Preview/Upload/Test).

   The hello-world.html file must be in the p1 folder on the server.

   For assistance, see Help Hours on the Syllabus in Canvas.

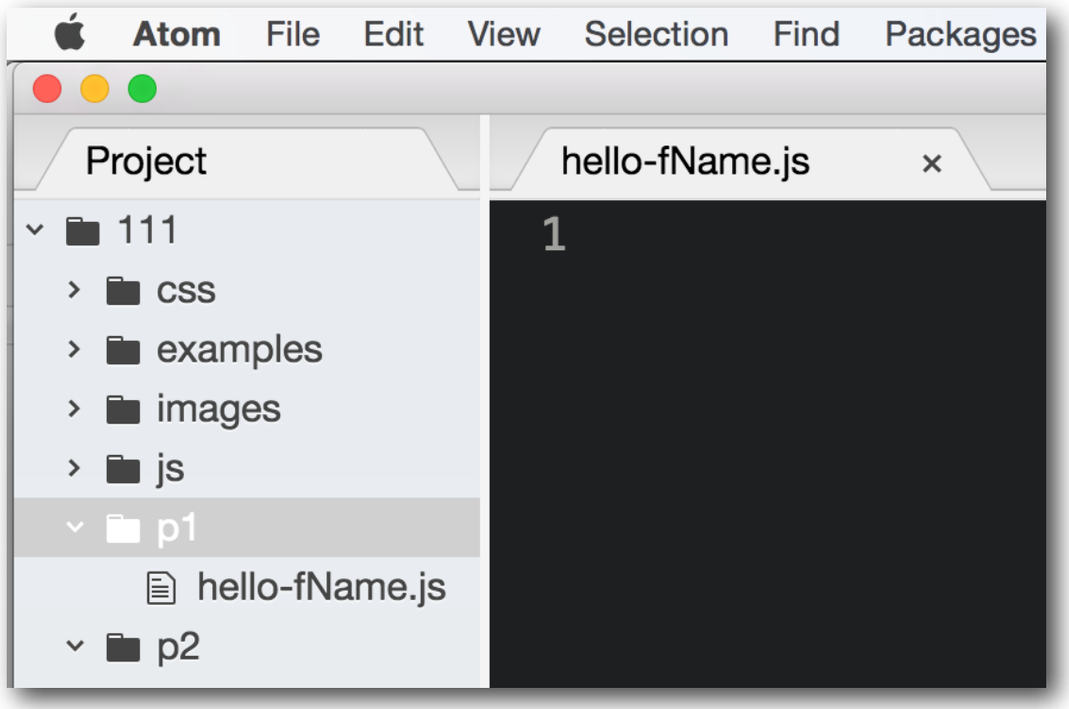
   Criteria for Success. You have met this requirement when:

   a) One file with exactly this file name is in your p1 folder on the uoregon.edu server: hello-world.html.
   b) When the page loads in Chrome, an alert displays the message, "Hello, World!". The same message is logged on the JavaScript console.
   c) hello-world.html contains the HTML elements described in steps 2a - 2f of the "Hello, World in JavaScript" section of the Lab Week 1 document.
   c) Your 111 folder on uoregon.edu is correctly set up for .htAccess password protection. When a user opens up your Jen's Kitchen web page for the first time, the user will be prompted for the .htAccess credentials:
   User name: 111
   Password: the middle 3 digits of your UO ID.

2. **[25 pts] hello-fName.js** This is an exercise in Command-Line JavaScript. Therefore, no web page (.html) is needed for this requirement.

   a) Start Atom, and open your 111 folder. To open a folder, choose the menu item *File* > *Open* (or *Open Folder* in Windows), and select your 111 folder from the dialog.

   When you open a folder in Atom, you will automatically get a Tree View on the left side of your window:

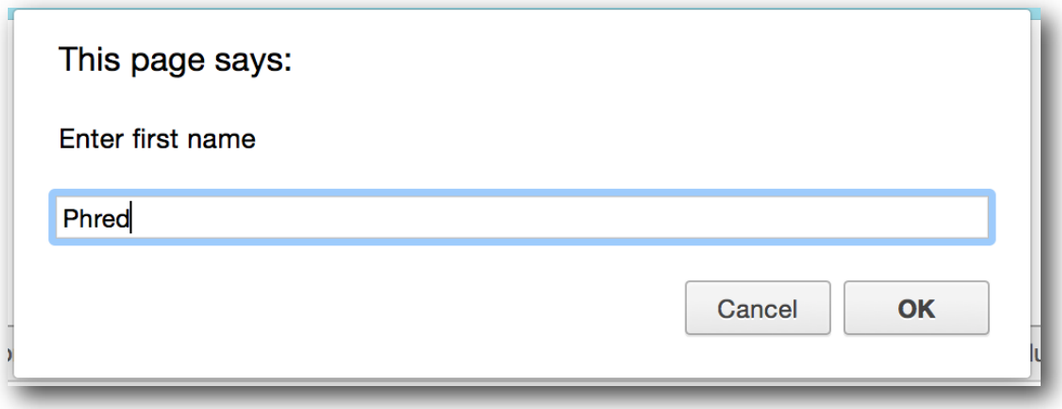


b) Create a new file name *hello-fName.js* (right-click the p1 folder, and choose New File. Name the file *hello-fname.js*.

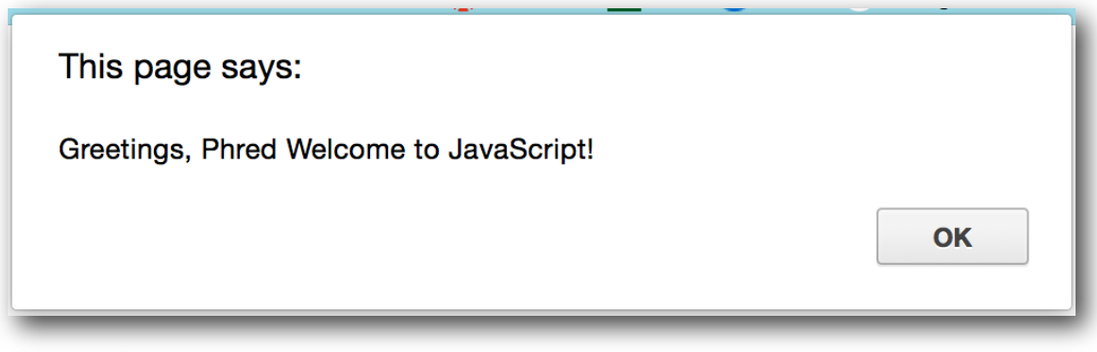
c) Read about the prompt function. Also read about String operators (+).

d) Add the JavaScript statements in *hello-fName.js* to prompt the user for their first name, store the value in a variable and display a greeting in an alert, and also using the console.log function. Examples:

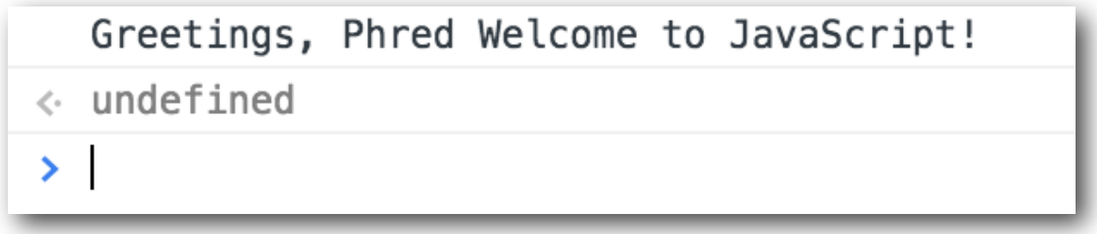
Prompt:



Alert:



Console:



When your JavaScript program is complete and correct (but not before), use Fugu or Putty to upload the .js file to your p1 folder on the server. Confirm that the .js file is on the server by opening it in Chrome. The JavaScript code will not execute, but this is OK-- it will just open as a text file in the browser. **Criteria for Success**. You have met this requirement when:

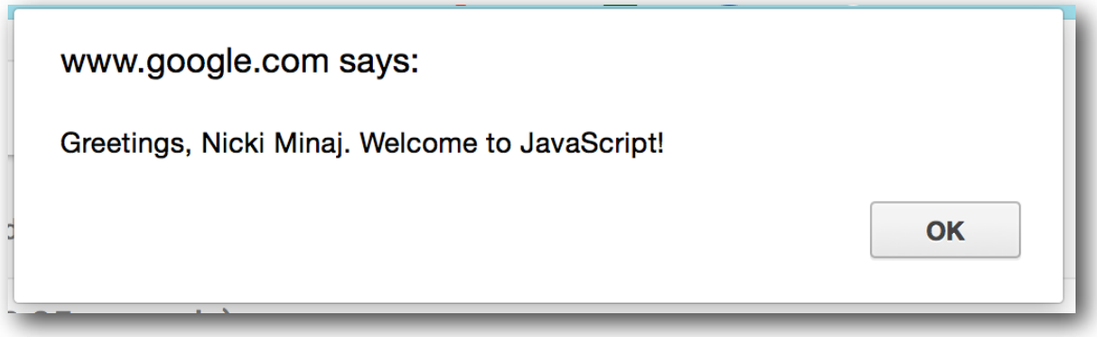
a) One file with exactly this file name is in your p1 folder on the uoregon.edu server: hello-fName.js.
b) hello-fName.js contains the JavaScript code to prompt the user for their first name, store the value in a variable and display a greeting in an alert, and also using the console.log function.
c) Your 111 folder on uoregon.edu is correctly set up for .htAccess password protection, as described in requirement 1, above.

3. **[25 pts] hello-fullName.js**. This is an exercise in Command-Line JavaScript.

a) In Atom, use the Save As command to save *hello-fname.js* as *hello-fullName.js* in your p1 folder.

b) Modify *hello-fullName.js* so that it prompts for the user's first and last name, and then displays a greeting with the user's full name. You will need to call the prompt function twice, and store each name in a different variable.

Example:



Display the message in both an alert and the console.

When your JavaScript program is complete and correct (but not before), use Fugu or Putty to upload the .js file to your p1 folder on the server. Confirm that the .js file is on the server by opening it in Chrome. The JavaScript code will not execute, but this is OK-- it will just open as a text file in the browser.

Criteria for Success. You have met this requirement when:

a) One file with exactly this file name is in your p1 folder on the uoregon.edu server: hello-fullName.js.
b) hello-fName.js contains the JavaScript code to prompt for the user's first and last name, and then display a greeting with the user's full name.
c) Your 111 folder on uoregon.edu is correctly set up for .htAccess password protection, as described in requirement 1, above.

4. **[25 pts]** Create a Responsive Image Gallery.

A) Add six image files to your 111/images/ folder. They can be images of anything you choose for a gallery.

B) Download responsive.html and responsive.css from Github. Store the two files in your 111/p1/ folder. Open responsive.html in Chrome to make sure the CSS is working.

C) To create a Responsive Image Gallery with the six images in your *p5/images/* folder:

i) **Add an image tag inside of each of the grid items**.

```
<div><img src="../images/oneOfyourImages.jpg"/></
div>
```

Reload the web page in Chrome, and to make sure the images are displayed.

ii) **Make the images fit into the grid items**. In this step, you will style the *img* element to fill the grid item, and then use *object-fit: cover.*

This will make the image cover its entire container, and the browser will crop it as needed.

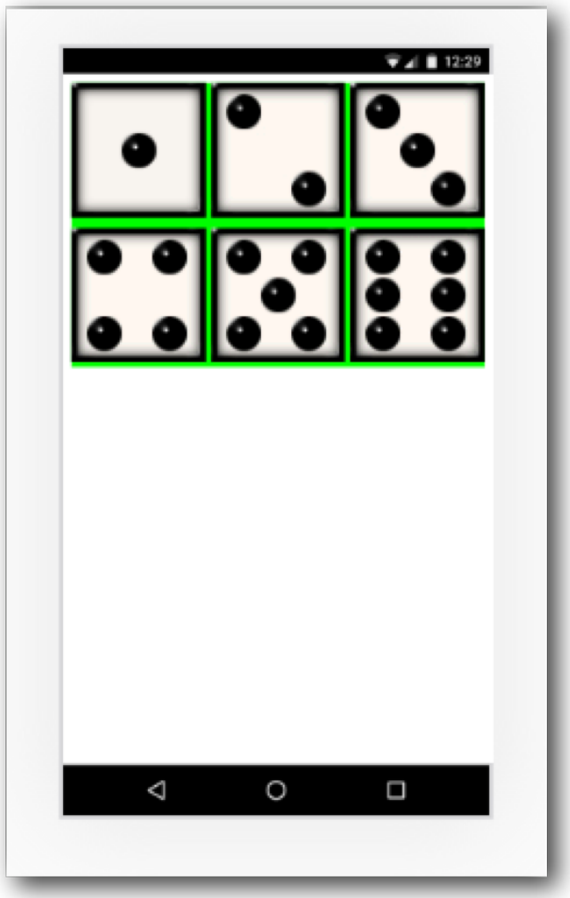
```
.container > div > img {
   width: 100%;
   height: 100%;
   object-fit: cover;
}
```

Open the web page in Chrome, and (A) drag the browser window to different sizes to see how the grid renders; (B) use the Device Toolbar (Chrome DevTools includes a Device Mode) to see how the grid behaves using different screen sizes and resolutions.

iii) Depending on the kind of images you are using, you may need to experiment with the size of your grid rows, as well. For example, I used the repeat() function, and did not specify a size:

```
grid-template-rows: repeat(2);
```

This is how my image gallery renders on a mobile device:



iv) When your responsive gallery is complete and correct (but not before), use Fugu or WinSCP to upload the .html, the .css and all six image files to their correct locations in your 111 folder on the server. Test your gallery on the server by opening it in Chrome.

CHECKPOINTS FOR FULL CREDIT

- Double-check that your files on the server meet the success criteria for each project requirement.
- Remember: It is not the files on your computer that are graded--it is the files on the server.
- Study How to Turn in your Project, below.
- Study Project Grading Checkpoints, below. Master these points, to get full credit on your projects.
- Study the 110 WebDev Workflow, below. Master these four steps to get full credit on your projects.

How to Turn In your Project

How to Turn In your Project

<u>All you Have to Do</u> is Make Sure your web pages are uploaded to the server and *tested* on the server by the Due-Date.

When your web pages are on the server, they can be graded.

You do not have to submit this project in Canvas, nor do you have to notify your instructor in any way (not even by Owl post).

Just make sure you complete the project by the Due-Date, and *do not upload or edit the files after the due-date*. If you change the web page files in any way after the due-date, this will change the time-stamp of the files on the server, and your project will be late (zero points).

Questions About this Project?

Do not send email to your instructors to ask questions about this project. Post your questions on Piazza, so all students in class can see the answer.

111 Help Hours in B004 PSC

See Help Hours on our Syllabus in Canvas.

Project Grading Checkpoints

HOW YOUR PROJECTS WILL BE GRADED

These checkpoints will help you get full credit on your projects.

- **The files you upload to the server by the due-date are what will be graded**, so be sure to test your web pages on the server to make sure they are correct.
- **Your job**: make sure the files are on the server on time, and that you have tested them to make sure they are correct.
- **There are no second chances.** We do not have the time or the resources to grade your work twice. Therefore make sure that *your website on the server* is correct. It is what is on the server that gets evaluated. Therefore, test your web pages on the server using Chrome (and nothing else) after uploading them.
- **Time-Stamps are Crucial.** When you upload a file to the server, it is stamped with the exact time of the upload. This time-stamp must be no later than the project due-date. Your project is on-time only if the time-stamps show that it was uploaded to the server on time.
- **Do not re-upload any of your project files after the due-date.** If you do, this will change the time-stamp and your project will be late (0 pts).
- If you are using Sublime Text, do not use Sublime's Sync button, as this can change the time-stamp on all your files on the server.
- **Your 111 folder on the server must be .htaccess password-protected.** If it is not, your project score will be zero (0). See your instructor or GTF for assistance (see Help Hours on our Syllabus in Canvas).

- **Know the 111 Late Policy as stated on the syllabus.**

The 111 WebDev Workflow

Here is the CIS 111 Web Development Workflow. Memorize these 4 steps.

1. **Edit**. Use a code editor (Atom, Sublime, or Brackets) to create a web page (.html and .js files) on your computer.

2. **Preview**. Open the web page on your computer using Chrome. When it is perfect, and not before, go to the next step.

3. **Upload**. Move all project files (.html, .js, .png, .jpg, ...) to the server using Fugu or Putty. This is also known as *Publishing* or *Deploying* the web page.

4. **Test**. Use Chrome to open your web page that is on the server. Do not use any other app-- use Chrome only. Your web pages will be assessed using Chrome.

   Type **http://pages.uoregon.edu/*yourDuckID*/111/** into Chrome to open the web page that is on the server. Use your actual DuckID in the URL.

   Make sure that this web page is correct, because that is what will be graded.

IMPORTANT: Read Beware the Browser Cache.

Related Glossary Terms

Drag related terms here

Index Find Term